1 多変量データ

1.1 作業ディレクトリ

作業ディレクトリ

```
> getwd()      # 現在の作業ディレクトリを表示
[1] "/Users/sakamoto/Documents/Rhome"

> setwd("/Users/sakamoto/Documents/AppStatB/rworking")
        # 作業フォルダを /Users/sakamoto/Documents/AppStatB/rworking に変更

> list.files()    # 作業ディレクトリ内のファイル一覧表示
[1] "galapagos.csv"           "gomi.csv"                "livestockmarket-j.csv"
[4] "livestockmarket.csv"     "tyuu2dannsitairyoku.csv" "中 2 成績.csv"
```

1.2 データの読み込み

データの読み込み

```
> gomi<-read.csv("gomi.csv")
       # csv 形式のデータ (gomi.csv) を読み込み，オブジクト gomi に代入
> ls()        # オブジェクトの一覧
[1] "gomi"
> gomi      # オブジェクト gomi の内容の確認
   地区番号 世帯数 排出量
1         1     73     37
2         2     63     27
3         3     31     18
4         4     24     11
5         5     79     39
6         6     84     40
7         7     32     14
8         8     33     18
9         9     66     28
10       10     36     17
       ###  このような形式のオブジェクトをデータフレームという．###
> names(gomi)   # データフレームの変数名の表示
[1] "地区番号" "世帯数"   "排出量"
> ncol(gomi)   # データフレーム gomi の列数
[1] 3
> nrow(gomi)   # データフレーム gomi の行数
[1] 10
```

1.3 データフレームの要素

データフレームの要素へのアクセス

```
> gomi[3,2]  # gomi の 3 行 2 列のデータ
[1] 31
> gomi[,2]   # gomi の第 2 列のデータ (変数名が世帯数のデータ)
 [1] 73 63 31 24 79 84 32 33 66 36
> gomi$世帯数  # gomi の変数"世帯数" のデータ (これは上と同じ)
 [1] 73 63 31 24 79 84 32 33 66 36
> gomi[,c(1,3)]   # gomi の第 1 列と第 3 列のデータ
   地区番号 排出量
1         1     37
2         2     27
3         3     18
4         4     11
5         5     39
6         6     40
7         7     14
8         8     18
9         9     28
10       10     17
> gomi[5,]   # gomi の第 5 行のデータ
  地区番号 世帯数 排出量
5        5     79     39
> gomi[c(3,6),]    # gomi の第 3 行と第 6 行のデータ
  地区番号 世帯数 排出量
3        3     31     18
6        6     84     40
> gomi[gomi$世帯数>=70,]   # 世帯数が 70 以上のデータ
  地区番号 世帯数 排出量
1        1     73     37
5        5     79     39
6        6     84     40
> gomi[gomi$世帯数>=70,3]    # 世帯数が 70 以上の排出量
[1] 37 39 40
> gomi[gomi$世帯数>=70,]$排出量     # これは上と同じ
[1] 37 39 40
> subset(gomi, 世帯数>=70)    # 世帯数が 70 以上のデータ
  地区番号 世帯数 排出量
1        1     73     37
5        5     79     39
6        6     84     40
> subset(gomi, 世帯数>=70&排出量<40)    # 世帯数が 70 以上で排出量が 40 以下のデータ
  地区番号 世帯数 排出量
1        1     73     37
5        5     79     39
```

1.4 データの単純集計

データの単純集計

```
> n<-nrow(gomi)                              # データ数を取得して，n に代入
> n                                          # データ数の確認
[1] 10
> mean(gomi$世帯数)                          # 世帯数の平均を求める
[1] 52.1
> var(gomi$世帯数)*(n-1)/n                   # 標本分散を求める．var は標本不偏分散
[1] 475.29
> sqrt(var(gomi$世帯数)*(n-1)/n)             # 標準偏差を求める
[1] 21.80115
> var(gomi$世帯数,gomi$排出量)*(n-1)/n  # 共分散を求める
[1] 219.41
> cor(gomi$世帯数,gomi$排出量)               # 相関係数を求める
[1] 0.9789491
> plot(排出量~世帯数,data=gomi)              # 散布図の描画
> gomi.lm<-lm(排出量~世帯数,data=gomi)  # 最小 2 乗法で回帰直線を求める
> coefficients(gomi.lm)              # 回帰係数を表示する
(Intercept)       世帯数
  0.8488712   0.4616339
> abline(gomi.lm)                            # 回帰直線を散布図に書き加える
```

1.5 分散共分散行列

分散共分散行列

```
> tairyoku<-read.csv("tairyoku.csv")         # 体力データを読み込む
> n<-nrow(tairyoku)      # 体力データのデータ数を n に代入する
> apply(tairyoku,2,mean)    # 体力データの列平均を求める (第 2 引数の 2 は列の意味)
     遠投       握力       身長       体重
 26.20000  33.33333 156.60000  47.40000
> v.tairyoku<-var(tairyoku)*(n-1)/n
            # 体力データの分散共分散行列を求める．var は不偏分散
          遠投       握力       身長       体重
遠投 15.22667 19.66667 18.68000 26.98667
握力 19.66667 45.42222 24.93333 50.06667
身長 18.68000 24.93333 48.77333 42.56000
体重 26.98667 50.06667 42.56000 77.04000
> v.tairyoku[2,3]  # 分散共分散行列の 2 行 3 列 (握力と身長の共分散) を求める
[1] 24.93333
> r.tairyoku<-cor(tairyoku)    # 体力データの相関行列を求める
           遠投       握力       身長       体重
遠投 1.0000000 0.7478150 0.6854618 0.7879319
握力 0.7478150 1.0000000 0.5297304 0.8463624
身長 0.6854618 0.5297304 1.0000000 0.6943082
体重 0.7879319 0.8463624 0.6943082 1.0000000
```

簡単な行列計算

```
> a<-c(1,2,3,4)   # c はベクトルを作る関数
> a
[1] 1 2 3 4
> amat<-matrix(a,nrow=2,byrow=TRUE) # matrix はベクトルを行列にする関数
                 # byrow=TRUE で 1 行目から行単位で並べる
> amat
     [,1] [,2]
[1,]    1    2
[2,]    3    4
> bmat<-matrix(c(3,2,1,0),nrow=2)
> bmat
     [,1] [,2]
[1,]    3    1
[2,]    2    0
> amat*2-3*bmat  # 行列の定数倍 (スカラー倍) と加法
     [,1] [,2]
[1,]   -7    1
[2,]    0    8
> amat%*%bmat  # 行列の積は %*%
     [,1] [,2]
[1,]    7    1
[2,]   17    3
> cvec<-c(3,5)
> amat%*%cvec # 行列とベクトルの積も %*%
     [,1]
[1,]   13
[2,]   29
> cvec%*%amat
     [,1] [,2]
[1,]   18   26
> cmat<-diag(cvec) # ベクトルを対角成分にする対角行列
> cmat
     [,1] [,2]
[1,]    3    0
[2,]    0    5
> t(amat)   # 転置行列
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> solve(amat) # 逆行列
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
> a.eigen<-eigen(amat) # 固有値, 固有ベクトルは関数 eigen
> a.eigen$values # 固有値は 変数名 values にある
[1]  5.3722813 -0.3722813
> a.eigen$vectors # 固有ベクトルを列ベクトルとする行列は vectors にある
           [,1]       [,2]
[1,] -0.4159736 -0.8245648
[2,] -0.9093767  0.5657675
> a.eigen$vectors%*%diag(a.eigen$values)%*%solve(a.eigen$vectors)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
```

分散共分散行列の性質

```
> t(v.tairyoku)-v.tairyoku  # 転置したものから分散共分散行列を引くと零行列になる
     遠投 握力 身長 体重
遠投    0    0    0    0
握力    0    0    0    0
身長    0    0    0    0
体重    0    0    0    0
> tai.eigen<-eigen(v.tairyoku)    # eigen は固有値と固有ベクトルを求める関数
> tai.eigen$values        # tai.eigen$values に固有値
[1] 150.924104  23.733838   7.540894   4.263386
> tai.eigen$vectors      # tai.eigen$vectors に固有ベクトル
           [,1]        [,2]       [,3]        [,4]
[1,] -0.2713301 -0.02029475 -0.2196303  0.93687281
[2,] -0.4873559  0.50255157 -0.6552450 -0.28386646
[3,] -0.4571831 -0.83889665 -0.2161757 -0.20125619
[4,] -0.6927105  0.20804438  0.6896989 -0.03442539
> tai.eigen$vectors[,1]%*%tai.eigen$vectors[,2]
                # 1 番目と 2 番目の固有ベクトルの内積
              [,1]
[1,] -1.110223e-16     # e-16 は 10 のマイナス 16 乗の意味，すごく小さい
> tai.eigen$vectors[,1]%*%tai.eigen$vectors[,1] # 1 番目の固有ベクトルの大きさ
     [,1]
[1,]    1
>
```

散布図行列

```
> pairs(tairyoku)     # pairs は散布図行列を描く関数
> install.packages("psych", dep=TRUE)    # psych というパッケージをインストールする
> library(psych)    #  psych というインストールされたパッケージを読み込む
> pairs.panels(tairyoku,smooth=FALSE,density=FALSE,ellipses=FALSE,
  scale=TRUE,pch=1,lm=TRUE)
    # pairs.panels は psych の中にある関数. pairs の少し多機能版
```

2 回帰分析

2.1 最小 2 乗推定量

最小 2 乗推定量

```
> ss<-var(tairyoku[,c(2,3,4)])*(n-1)/n
       # データフレーム tairyoku の 2〜4 列の分散共分散行列
> ss
         握力      身長      体重
握力 45.42222 24.93333 50.06667
身長 24.93333 48.77333 42.56000
体重 50.06667 42.56000 77.04000
> s<-var(tairyoku[,1],tairyoku[,c(2,3,4)])*(n-1)/n
       # データフレーム tairyoku の 1 列めと 2 〜 4 列の共分散
> s     # s は行列
         握力  身長      体重
[1,] 19.66667 18.68 26.98667
> bhat<-solve(ss)%*%t(s)  # 回帰係数を求める
> bhat
          [,1]
握力 0.2013769
身長 0.1710246
体重 0.1249428
> heikin<-apply(tairyoku,2,mean)   # データフレーム tairyoku の列単位の平均
> heikin
       No       遠投       握力       身長       体重
  8.00000  26.20000  33.33333 156.60000  47.40000
> xbar<-heikin[c(3,4,5)]  # データフレーム tairyoku の 3, 4, 5 各列の平均
> ybar<-heikin[2] # データフレーム tairyoku の 2 列の平均
> bhat0<-ybar-xbar%*%bhat     # 回帰係数の切片を求める
> bhat0
         [,1]
[1,] -13.2173
> tairyoku.lm<-lm(遠投~握力+身長+体重,data=tairyoku)
               # 関数 lm で最小 2 乗推定量を求める
> tairyoku.lm$coefficients
(Intercept)        握力        身長        体重
-13.2172983   0.2013769   0.1710246   0.1249428
```

2.2 残差・予測値・寄与率

残差・予測値

```
> pred<-as.matrix(tairyoku[,c(3,4,5)])%*%bhat+as.numeric(bhat0)
          # 回帰係数の推定量 bhat とデータフレーム tairyoku から予測値を求める
> pred      # 予測値
          [,1]
 [1,] 21.63890
   ............
   ............
[15,] 20.10078
> resid<-as.matrix(tairyoku[,1])-pred   # 実測値から予測値を引いて，残差を求める
```

平方和・寄与率

```
> see<-var(resid)*(n-1) # 残差平方和
> syy<-var(tairyoku[,1])*(n-1)  # 遠投の偏差平方和
> srr<-var(pred)*(n-1)  # 予測値の偏差平方和 (回帰によるばらつき)
> srr+see
      [,1]
[1,] 228.4
> syy    # syy=srr+see(平方和の分解)
[1] 228.4
> r<-cor(tairyoku[,1],pred) # 目的変数と予測値の相関 (重相関)
> r2<-r^2   # 重相関の 2 乗＝寄与率（決定係数）
> r2
         [,1]
[1,] 0.691349
> 1-see/syy  # 寄与率を平方和で計算する方法
         [,1]
[1,] 0.691349
> rs2<-1-(see/(n-3-1))/(syy/(n-1)) # 自由度調整済み寄与率
> rs2
          [,1]
[1,] 0.6071714
> sighat<-see/(n-3-1) # 分散の不偏推定量
> sighat
         [,1]
[1,] 6.408718
```

lm 関数の要約

```
> tairyoku.lmsum<-summary(tairyoku.lm)
       # tairyoku を最小２乗法で回帰係数を計算するときの諸量の集計
> names(tairyoku.lmsum)  # オブジェクト tairyoku.lmsum の変数一覧
 [1] "call"          "terms"         "residuals"     "coefficients"  "aliased"
 [6] "sigma"         "df"            "r.squared"     "adj.r.squared"
[10] "fstatistic"    "cov.unscaled"
> tairyoku.lmsum$r.squared
         # tairyoku.lmsum の変数 r.suquared に決定係数が格納されいる
[1] 0.691349
> tairyoku.lmsum$adj.r.squared
    # tairyoku.lmsum の変数 adj.r.suquared に自由度調整済み決定係数が格納されいる
[1] 0.6071714
> tairyoku.lmsum$sigma^2
      # tairyoku.lmsum の変数 sigma に誤差の標準偏差の推定量が格納されている
[1] 6.408718
```

2.3 回帰係数の検定

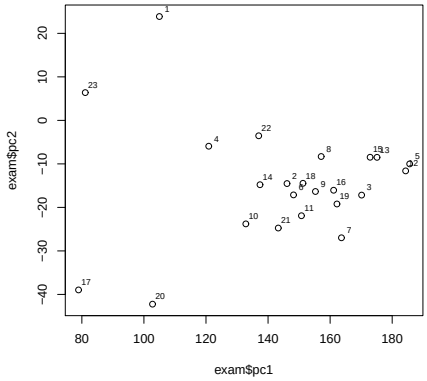
lm 関数の要約

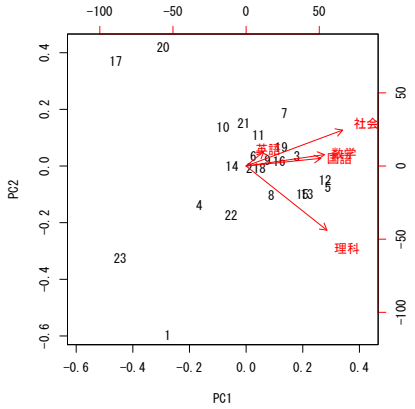
```
> (srr/3)/(see/(n-3-1))   # 回帰係数の検定の検定統計量
         [,1]
[1,] 8.212984
> qf(0.99,3,11)    # 自由度 (3,11) の下側９９％点 (上側１％点) … 棄却限界値
[1] 6.21673
> tairyoku.lmsum$fstatistic
               # 変数 fstatistic には，検定統計量の値と自由度が格納されている
    value     numdf     dendf
 8.212984  3.000000 11.000000
> 1-pf(tairyoku.lmsum$fstatistic[1],3,11)
          # 関数 pf はエフ分布の分布関数．したがって，1-pf は p 値．
      value
0.003773589
> anova(tairyoku.lm)
         # 各回帰係数がゼロであるか (帰無仮説)，そうでないか（対立仮説）の検定統計量
         # などをまとめた分散分析表.
         # Pr(>F) が p 値で，5% 以下で *，1% 以下で **，0.1%以下で ***
Analysis of Variance Table

Response: 遠投
          Df  Sum Sq Mean Sq F value    Pr(>F)
握力       1 127.727 127.727 19.9303 0.0009558 ***
身長       1  26.576  26.576  4.1469 0.0665153 .
体重       1   3.600   3.600  0.5618 0.4692729
Residuals 11  70.496   6.409
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

3 主成分分析

主成分分析

```
> exam<-read.csv("exam-j.csv")   #  試験データの読み込み
> exam
   No 国語 社会 数学 理科 英語
1   1   47   61   56   21   95
.............................................
.............................................
23 23   29   33   55   79   84
> n<-nrow(exam)      # 試験データのデータ数
> v.exam<-var(exam[,c(2,3,4,5,6)])*(n-1)/n   # 試験データの分散共分散行列
> v.exam
          国語       社会       数学       理科      英語
国語 239.93951 193.69754 168.14745 156.13422 45.46314
社会 193.69754 360.66163 214.43289 178.06238 59.92439
数学 168.14745 214.43289 239.24386 163.34405 53.01512
理科 156.13422 178.06238 163.34405 326.82042 14.28355
英語  45.46314  59.92439  53.01512  14.28355 24.95274
> eigen.exam<-eigen(v.exam)   # 試験データの固有値と固有ベクトルを計算
> eigen.exam$values    # 試験データの固有値
[1] 845.030828 176.209851  95.166066  67.023864   8.187538
> sum(eigen.exam$values[c(1,2)])/sum(eigen.exam$values)
             # 第 2 主成分までの累積寄与率
[1] 0.8570201
# 第 1 主成分から第 2 種成分までをデータフレームに変数名 pc1, pc2, pc3 で格納
> exam$pc1<--as.matrix(exam[,c(2,3,4,5,6)])%*%eigen.exam$vectors[,1]
> exam$pc2<--as.matrix(exam[,c(2,3,4,5,6)])%*%eigen.exam$vectors[,2]
> var(exam[,c(7,8)])
         # 主成分の分散共分散行列．共分散が 0 で，分散が固有値と一致する
             pc1          pc2
pc1 8.834413e+02 1.451364e-14
pc2 1.451364e-14 1.842194e+02
> plot(exam$pc1,exam$pc2)  # 第 1 主成分と第 2 主成分のグラフ (下図)
> text(exam$pc1+xinch(0.1),exam$pc2+yinch(0.1),exam$No,cex=0.7)
     # グラフにデータラベルを付加
```





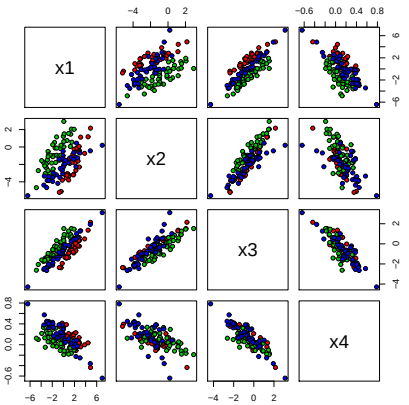
prcomp で主成分分析

```
> pc.exam<-prcomp(exam[,c(2,3,4,5,6)])  # prcomp は主成分分席を行う関数
> pc.exam
Standard deviations:   # 固有値の平方根
[1] 29.722741 13.572744  9.974557  8.370807  2.925696
Rotation:   # 固有ベクトル
            PC1        PC2         PC3         PC4         PC5
国語 0.4481043 -0.1012605  0.75339195 -0.46321069  0.08237283
社会 0.5777951 -0.4712322 -0.59458373 -0.29255806  0.07051953
数学 0.4687250 -0.1494315  0.18924463  0.82794928  0.19145111
理科 0.4842098  0.8421800 -0.18941761 -0.04770349 -0.13460715
英語 0.1057972 -0.1899728  0.08472421  0.10986357 -0.96616205
> summary(pc.exam)   # prcomp による主成分分析の結果 pc.exam を要約
Importance of components:
                          PC1     PC2     PC3     PC4     PC5
Standard deviation     29.7227 13.5727 9.97456 8.37081 2.92570
Proportion of Variance  0.7091  0.1479 0.07986 0.05625 0.00687
Cumulative Proportion   0.7091  0.8570 0.93688 0.99313 1.00000  #累積寄与率
> biplot(pc.exam) # 第 1 主成分と第 2 主成分の散布図 (軸は偏差)
   # 各教科の重みをベクトルで表現してある.
```

4 判別分析

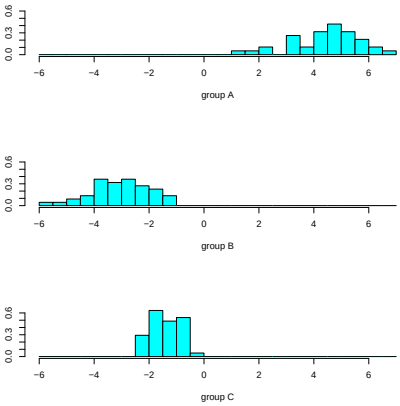
判別分析 (マハラノビス距離による)

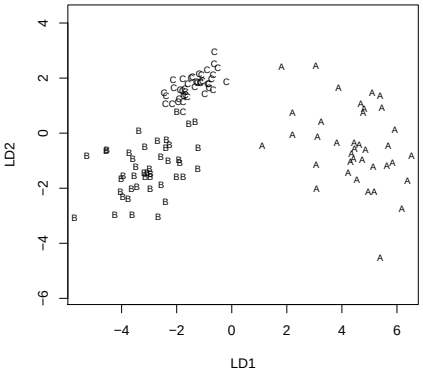
```
> dexam<-read.csv("daca-ex.csv")
> pairs(dexam[1:4],pch=21,bg=c("red","green3","blue")[unclass(dexam$g)])
     # グループで色分けして，散布図行列を描く
> m.dexam<-by(dexam[,c(1,2,3,4)],dexam$g,colMeans)
     # 変数 g でグループ分けして，平均を求めて，m.dexam に格納する.
> mlevar<-function(x) var(x)*(nrow(x)-1)/nrow(x)
     # 標本分散を計算する関数 mlevar の定義
> v.dexam<-by(dexam[,c(1,2,3,4)],dexam$g,mlevar)
              # v.dexam には 変数 g の値でグループ分けして，分散共分散行列を求めている.

> y1<-c(1,0,-1,1)     # 判別するデータ y1 を定義
> mahalanobis(y1,m.dexam$A,v.dexam$A)
[1] 115.728       # y1 とグループ A のマハラノビス距離
> mahalanobis(y1,m.dexam$B,v.dexam$B)
[1] 97.45129     # y1 とグループ B のマハラノビス距離
> mahalanobis(y1,m.dexam$C,v.dexam$C)
[1] 560.4156      # # y1 とグループ C のマハラノビス距離
     # y1 は B と最も近いので， B から出現したと考える.
> y2<-c(1,1,0,1)
     # これについてもマハラノビス距離で比較すると A のデータである
> y3<-c(-0.3,-2.4,-0.8,0.2)
    # これについてもマハラノビス距離で比較すると C のデータである
```



判別分析 (B-W 法による)

```
> library(MASS)   # B-W 法の関数 lda を含むライブラリ MASS  をロード
> lda.dexam<-lda(g~x1+x2+x3+x4,data=dexam)
      # lda.dexam に B-W 法による判別方法が格納される
      # g~x1+x2+x3+x4 はグループ g は変数 x1, x2, x3, x4 で決まるという
      # モデルの表現形
> lda.dexam
Call:
lda(g ~ x1 + x2 + x3 + x4, data = dexam)

Prior probabilities of groups:
        A         B         C
0.3089431 0.3577236 0.3333333
   # グループのデータの比率
Group means:
          x1         x2         x3         x4
A  1.8804652 -1.7491132 -0.3398922 0.09204994
B -1.3570611 -0.6428588 -0.4334403 0.07588114
C -0.2805611 -2.3897719 -0.7978859 0.19417596
     # グループごとの平均

Coefficients of linear discriminants:
          LD1        LD2
x1   3.632028 -0.9800165
x2   2.829613 -2.8711649
x3 -10.540114  6.8380946
x4  -9.812387 13.0875448
     # B-W 法による重み
Proportion of trace:  # 寄与率
   LD1    LD2
0.8588 0.1412
> plot(lda.dexam,dimen=1) # 第 1 の合成変量の分布
        # 群ごとに分布が分離していることが確認できる
> plot(lda.dexam,dimen=2) # 第 1 と第 2 の合成変量の 2 次元分布
        # 群ごとに分布が分離していることが確認できる
> lda.dexam$scaling
          LD1        LD2
x1   3.632028 -0.9800165
x2   2.829613 -2.8711649
x3 -10.540114  6.8380946
x4  -9.812387 13.0875448
> (t(lda.dexam$scaling)%*%(y1-m.dexam$A))^2
           [,1]
LD1  0.04004287
LD2 10.30304362
    # 第 1 の合成変量による距離
> apply((t(lda.dexam$scaling)%*%(y1-m.dexam$A))^2,2,sum)
[1] 10.34309   # y1 とグループ A の距離の 2 乗
               # 第 1 と第 2 の合成変量の距離の 2 乗和
> apply((t(lda.dexam$scaling)%*%(y1-m.dexam$B))^2,2,sum)
[1] 69.57348   # y1 とグループ B の距離の 2 乗
> apply((t(lda.dexam$scaling)%*%(y1-m.dexam$C))^2,2,sum)
[1] 32.86697   # y1 とグループ C の距離の 2 乗
# B-W 法ではグループ A が最も近く，y1 は A に属する
```


group A
group B
group C


LD1
LD2